Rich Representation and Visualisation of Time-Series Data

Simon Kerr, Greg Foster and Barry Irwin
Department of Computer Science and Department of Information Systems
Rhodes University, Grahamstown 6140
Tel: 046 6038291, fax: 046 6361915
e-mail: g03k0704@campus.ru.ac.za; g.foster@ru.ac.za; b.irwin@ru.ac.za

***Abstract* - Currently the majority of data is visualized using static graphs and tables. However, static graphs still leave much to be desired and provide only a small insight into trends and changes between values. We propose a move away from purely static representations of data towards a more fluid and understandable environment for data representation. This is achieved through the use of an application which animates time based data. Animating time based data allows one to see nuances within a dataset from a more comprehensive perspective. This is especially useful within the time based data rich telecommunications industry. The application comprises of two parts - the backend manages raw data which is then passed to the frontend for animation. A play function allows one to play through a time series. Which creates a fluid and dynamic environment for exploring data. Both the advantages and disadvantages of this approach are investigated and an application is introduced which can be used to animate and explore datasets.**

Index terms- Adobe Flash, data representation, visualisation, 2D animation, XML, graphical data, time-series data

I. INTRODUCTION

STATIC 2D graphs have long been one of the foremost ways of visualising data in an attempt to make more sense of it [1]. Plotting two or more types of values against one another allows trends and nuances within data to be more easily identified than would otherwise be apparent in a list or table. Telecoms data for instance is rich in trend based data which needs a large amount of exploration before it can be easily understood. The majority of datasets are recorded over a specific time period at varying time intervals. Unfortunately, a major disadvantage of static graphs is that they do not readily allow time-series data to be displayed with more than one other attribute (such as line graphs which can display time on one axis and another attribute on the other) on the same graph. Other forms of charting data (such as bubble charts) allow for points to be displayed with different sizes, allowing for further information to be gleaned from the visualisation. Analysis of time-series data can be greatly enhanced by mapping it to an animated timeline.

Graphical data visualisation is the next step in visualising trends and creating understandable interfaces for statistics users [2]. Data visualisation allows further attributes to be used (such as an extra axis) and hence the ability to visualise more than one attribute in a single visualisation. Various types of visualisations can be seen in the works of Heer and Card [3] and range from radial layouts (see Fig 1a) and tree based layouts (see Fig 1a and b) to spatial based charts (see Fig 1d,e and f).

A large body of work has come from a number of data visualization prototypes which have been released by *Gapminder* [4]. These tools are geared towards creating animated presentations of statistics. While these tools allow for highly informative presentations of statistics, they do not provide the user with the ability to dynamically enter their own data. Many different ways of representing data have been developed and tweaked to the kinds of data people wish to gain a greater understanding of. These include all the variants of 2-axis line graphs, charts such as bar and pie charts and various types of tables.

All of the aforementioned methods of viewing data have one thing in common and that is that they are all static in nature. Static graphs merely give a space orientated representation of one value against another. Static graphs (such as bar charts and line graphs) plot values against each other in various different spatial ways. Tables simply map data into columns and rows and leave the reader to discover the trends within the dataset. All of these use spatial based representations to portray their underlying meaning. Fluid data looks at adding additional ways in which to view datasets.

Fluid data is data which is not in a traditional format (i.e. static graphs and tables) and which enhances the underlying meaning of datasets by adding additional attributes (such as animation). Some examples of a move towards fluid data can be seen in [3]. Rosling *et al.* [2] have found that many people know very little about statistics in general, hence an emphasis is placed on the importance of making statistics understandable to a larger audience. The lack of a general understanding pertaining to statistics is a very important driving force in creating software to aid in this area. As a result, a movement away from purely static representations of data towards a more fluid and understandable environment is proposed. This is achieved through the use of animation applied to time-series data.

Ronnlund *et al.* [1] introduces a model which represents data by animating graphs using Adobe Flash. The model lays out the data in a conventional scatter plot (2 axes with points represented as circles) and animates and resizes the points as the time frame (in this case years) increments. However, this prototype does not allow one to insert ones' own data. As a

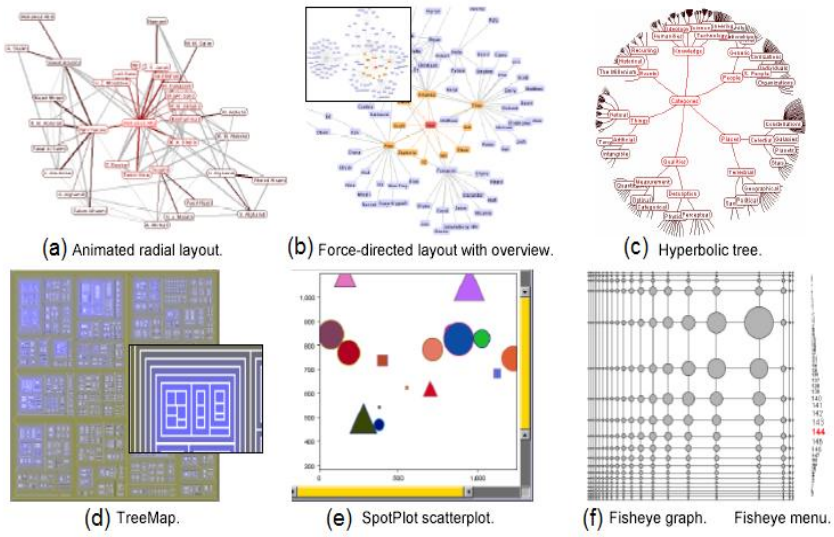

Fig 1. Examples of data visualisation layouts.

result of this, our research focused on an application that allows custom data to be dynamically animated in a 2D visualisation.

This paper presents an application which allows us to explore data sets in a similar way to that set out in [2], but which would be dynamic and accept any correctly formatted dataset. We wanted to move to a more fluid and dynamic environment, away from static representations of data which are cumbersome and potentially difficult to understand.

II. SUITABILITY OF ADOBE FLASH AS DEVELOPMENT ENVIRONMENT

Adobe Flash based applications have grown in popularity in the past few years [5]. Initially Flash was simply an animation tool for the Web. However, it has now become a versatile Integrated Development Environment (IDE) which can create powerful applications and animations (which run on the Flash player). The power of Flash is realised in its strong vector based frontend and object-orientated scripting language (ActionScript), thus making it a perfect tool for creating richly animated data visualisations.

The use of vector graphics (images created from underlying mathematical descriptions) is perfectly suited to this project. Plugging in numerical data values corresponds well to simply drawing and animating a frontend that is based on these values. Flash's ActionScript allows for object-oriented programming, provides an API for XML parsing and supports various auxiliary functions for easy access to and input of comma separated values (CSV) data [5]. It is these features that allow for a perfect coupling of dynamic frontend animation with a swift backend for data manipulation thus allowing for the creation of a fluid data visualisation application.

III. DESIGN AND IMPLEMENTATION

An object-orientated design within a vector graphics based environment allows for a dynamic graphing application to be built. Combining ideas from [2] and theory on creating visualisation techniques led to the creation and design of an application which could easily plot and simulate trends graphically. The following describes how this application was created and what choices were taken at the various steps of implementation.

In this study, Adobe Flash Professional CS3 was used and development took place within the Flash IDE. The implementation of the application was split into two main parts, namely the backend and the frontend. The backend contains all the data handling classes and functions. The frontend contains various drawing and geometric functions to interpret, scale and draw all relevant data passed to it from the backend.

We will first explore the backend of the application which includes the CSV manipulation and XML parsing. This will be followed by an overview of the frontend, including the various types of drawing and scaling functions used.

A. *Backend*

The backend of the application needs to be able to dynamically generate objects which the frontend can visualise. To this end the basic requirements are to:

1. Recognize correctly formatted data by reading in ordered CSV and making sure that the correct order is maintained (time, *x-coordinate*, *y-coordinate*, group or colour size)
2. Transform CSV into a more structured and portable format (e.g. XML) and pass this formatted data to the frontend.

The CSV is read in through a file reader and split into an array. This quickly builds up an array which contains all the data.

The layout of the XML is standardized and therefore it is a trivial matter of building up XML strings and wrapping them in their respective elements.

Once all the CSV is converted into a well-formed XML document, it is passed to the frontend which extracts each *point object* tag set (which contains all information to be drawn including the ordered pair values) and converts it into a *point object* object in code, similar to VisualItems [3], where each of its values can programmatically be accessed. The data transformation process is mapped out from the backend to the frontend as follows:

- *Step 1*: Creation of the CSV which contains time, *x* value, *y* value, group (colour) and size.
- *Step 2*: The CSV is formatted into XML which is passed to the frontend. XMLs main function is to provide a structure for sharing data and so is perfect for storing and transporting data within the application [7].
- *Step 3*: *point objects* are created which contain all the details given to it by the backend (via the XML file).
- *Step 4*: *drawn objects* are drawn. *drawn objects* are *point objects* which has been created in the main frontend user interface.

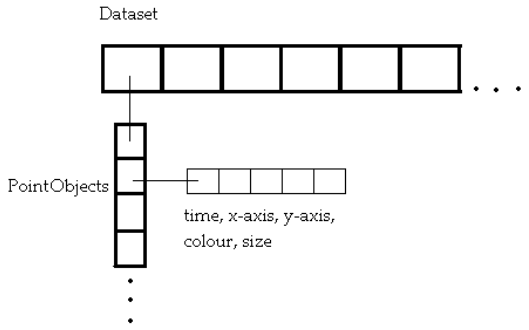

Fig 2. The main horizontal array contains each dataset. Within each dataset is an array of *point objects* (one point on the graph). Each *point object* contains time, *x*-axis value, *y*-axis value, colour and size values.

The backend extracts groups of five ordered values, marks them with opening and closing tags and encapsulates these five tags within a *point object* element. Once a delimiter is detected all previous *point object* elements are encapsulated within a *point objects* element (Fig 2). This represents the entire movement of the point object from the beginning to the end of the time values. Once the end of the array is reached the entire set of tags is encapsulated in the dataset tag so as to create a well formed structured XML file. This file is then passed to the frontend which assigns each *point object* within the XML document to a *point object* within code.

B. *Frontend*

The frontend creates a matrix of *point objects* and sets up the slider time values by applying a scaling function to the slider. The slider is used to quickly browse through the data and allows the drawing of connecting lines. A play function animates all data through the main time display. When the time reaches that of a *point object*, it is either drawn on the graphing area or is animated to its new position. The frontend needs to present a fluid interface to the user and in order to achieve this, these basic requirements are needed:

1. A bubble chart interface similar to [2] because it is the most adaptable type of static graph for animation.
2. Two axes, namely x-axis and *y-axis* for plotting data on the Cartesian plane.
3. A slider which allows the user to quickly browse through the data.
4. A play function, which shows smooth transitions from one time instance to the next.
5. The ability to connect plotted points with lines, highlighting various trends and allowing line graphs to be displayed.
6. A dynamically generated legend based on the inputted data the user provided.

Animation, size and colour can be used to good effect in conveying the meaning of data [2]. Adding up to three additional attributes (time, size and colour) to the static scatter plot allows as many as four properties (*x value*, *y value*, time and size) to be mapped and animated over a given time period (Fig 3) effectively creating an animated bubble chart. This creates a fluid environment which easily conveys the trends and movements in the input data. It would be possible to simply redraw all values at their new positions but this produces a somewhat jerked effect. This is adequate if one only wishes to jump to a particular region in the information but this loses aesthetic appeal as a fully animated fluid and dynamic environment.

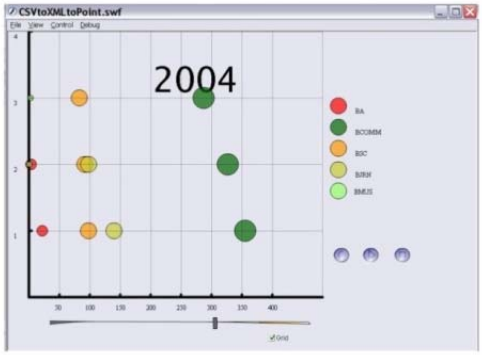

Fig 3. The frontend showing *drawn objects* with attributes displayed.

Moving to a fluid environment and applying animation to plotted points which can animate both position and size allows many datasets to be converted into this format and explored.

1) Frontend data handling: The frontend extracts each *point object* from the XML file and assigns each of its values to a created object. These *point objects* are stored in a matrix with a new array being created within the main array every time the *point objects* tag is closed in the XML. Each array of *point objects* corresponds to all possible positions for one ordered pair (Fig 4).

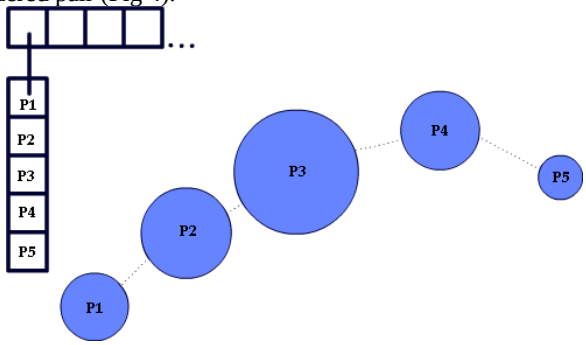

Fig 4. All *point objects* that belong to the same dataset contain different information at the progressive time instances. The mapping from P2 in data to P2 being visualised is shown by the arrow. Note that P1 – P5 is the same ordered pair but at different time intervals.

A simple iteration through the matrix allows every possible position to be calculated. To find out where the plotted point should move to from its initial point at P1, a query to *P2.xaxis* and *P2.yaxis* will return the coordinates and a query to *P2.size* will return the new size which must be drawn. The drawing of the *point objects* is discussed below.

C. Drawing functions

The main components that need to be drawn are the axes, the grid lines and trail lines as well as the plotted points. The drawing functions used by the application can be divided into three main groups: axis and line drawing functions, *drawn object* functions and auxiliary drawing functions.

1) Axis and line drawing: The axes are drawn as soon as the frontend is initialized. They are automatically drawn to create a 500 × 500 pixel square (the stage). Once the axes are drawn the gridlines and tick marks are dynamically generated from the data. The *x* and *y* axes are generated at a fixed origin (0,500). This means that scaling functions (see section 3.4) must be used in order to make all data either fit within the given area, or if the data values are small, expanded to take advantage of all the space available on the stage.

2) Drawn object functions: Although the placement of points falls under the animation functions, the initial drawing of objects uses various drawing functions to set up the data points (circles). The points are drawn using a circle drawing function. In Flash a movieClip is a data type and specifies an object on the stage which can contain any graphical element. In the case of this application, each *point object* generates one movieClip which is initially saved within an array. The movieClip can be transformed, manipulated and moved around the interface and this creates all the frontend animation. When a function is called to draw a new *point object*, a new movieClip is created and positioned and the circle is drawn within that movieClip.

A new array is created which generates a moveClip for each dataset. This movieClip is then stored in the newly created array and its *x* and *y* values are set to the initial values specified in the matrix by the values passed in from the backend. The stored movieClip is part of an object called a *circle Object*. This object is created to hold the movieClip, *x-coordinate*, *y-coordinate* and size values in each address in the new array. Because of the nature of movieClips (they are separate instances arranged in a tree structure), they can simply be moved around the stage as required by the data. This is much more efficient than redrawing the frames between movements to new points on the graph or creating new objects for every different time value of each *point object*.

3) Auxiliary drawing functions: The auxiliary drawing functions are functions which contribute to the overall layout, such as the scaling of values and the offsets applied to everything on the stage. These functions are also involved in creating the trail lines (to create line graphs) by drawing a line from each data point at each time instance. In addition, the trail lines can be enabled when dragging the slider.

D. Scaling

All values are scaled so as to fit into the stage. Scaling is achieved by multiplying all values by the scaling ratio, (500/maximum value in dataset). Tick labels for ordered pairs are dynamically generated, based on the maximum value in the dataset (larger tick intervals are required for datasets with higher values and smaller tick intervals for datasets with lower values) by applying the scaling ratio which was applied to all of the ordered pairs in the dataset. For example, if the tick interval number is 50 units and the dataset has been scaled then the label at the tick interval would be the actual co-ordinate position of 50 × (500/M).

1) Slider scaling: The slider is used to quickly scroll through the dataset. It allows the user to browse through time and draws all *point objects* which corresponds to that specific time frame. Similarly, the slider is scaled in the same way as the data. The slider works when the user clicks and drags the slider thumb which results in the time display being updated to reflect the new time.

Scaling the slider values allows the slider to display only the relevant information to a particular time frame This means that only certain ranges of the slider's *x* value pertain to a single time value within the data, this works well and looks as though the slider is 'snapping' to each new time value. The slider is also the main interface for drawing trail lines which show the change in each *drawn object* over time as a line graph.

2) Trail lines: The trail lines are drawn as the slider is moved along. The lines are drawn at each successive change in a *drawn object*'s *x* and *y-coordinate* value and are of the same colour as the *drawn object* they trail behind (Fig 5). The advantage of these trail lines is that they allow the seamless conversion from bubble chart to line graph, further assisting in interpreting data trends.

Fig 5. The slider thumb at maximum value showing all points at all time values. All trail lines are shown along with a legend on the right and gridlines behind the *drawn objects*.

E. Frontend Animation

There are two different types of animation which take place in the application. The first is initiated when the slider is dragged. This initiates each *point object* to be drawn at its time position and is immediately changed when the slider is dragged again. This allows the user to browse through the data at their own pace. The second is initiated by the play button and animate the data over a progressing time value.

1) Play function: When animating the data with the play button there are one of two actions which must occur within the application. If the current *point object* has not been used on the stage, then it must be faded in at its appropriate position. However, if the *point object* has been drawn and the *point object's* next time value is at the current time displayed, it is animated to the new position.

IV. EVALUATION AND DISCUSSION

An evaluation of how the application copes with real world data was explored and expected performance versus

real performance recorded. The test data used was that of registered Internet users per continent. The data is fairly large (ranging from a few million to over a billion) and hence proved to be useful in determining performance of a typical real world datasets. A user evaluation (qualitative feedback from 10 users) was also conducted in which valuable feedback was attained about how the system performs compared to static graphs. Users were shown data compiled into static graphs and asked to answer a few questions about the nature of the data. The users were then asked to look at the same data within the visualisation software and their views on the ease of use and ease of understanding were gathered.

A. *Evaluation Data*

The data from the different continents were entered into a separate CSV file. Once the data was converted into ordered CSV it was loaded into the application. The data was mapped so that continental population was mapped to the y-axis and registered Internet users to the x-axis. Size also represented the number of Internet registrations, while colour was used to differentiate between continents.

Fig 6 represents the backend data screen. On the left hand side is the XML output which displays all *point objects* within their respective tags. While on the right hand side there is a text based display of each *point object* allowing the user to browse through the data in its *point object* form. Once this backend display has been viewed the user can press the forward button and view the initial frontend screen. The initial frontend screen is created displaying the first dataset representing overall registered users (Fig 7). All initial values at the first time instance are displayed and the slider thumb is set to its initial value.

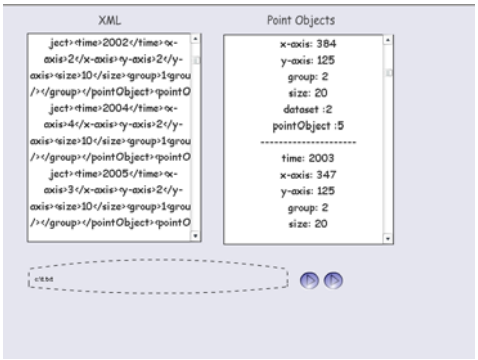

Fig 6. The backend and frontend data display, displaying XML on the left and *point objects* on the right. The buttons on the bottom right allow the user to progress to the frontend initial screen.

If the user drags the slider, a quick view of all the data is shown. If the 'lines' checkbox is selected, then trail lines joining previous points on the graph are drawn. These trail lines provide a means of displaying a line graph, however due to the nature of most time-series data (and bubble charts) a line graph can quickly become cluttered (Fig 5). However, this is not the case of all data and the option to do this is still a useful one for tracking various movements within the data.

Pressing the play button disables all trail lines and resets the slider (Fig 7). All initial *drawn objects* are faded in at the first time instance and as time progresses new *drawn objects* are faded in to the current time value.

Fig 7. The final positions of the data reached by the play function. The *y*-axis corresponds to continent population and the *x*-axis to the registered Internet users (units are in millions). Size is also tied to Internet users and hence grows as the time progresses.

The visualisation animates as time passes every two seconds. This value can be changed to speed up or slow down the animation. The visualisation can be paused at any time and the values read using the grid. The application performed seamlessly using real world data. Because of the size difference of registered users within each continent it was possible to give a *drawn object* size values which corresponded to the Internet registration count This added an additional attribute which was not present in the static data (tables) but which was directly related to the *x*-axis. Coupled with colour which is mapped to continent, it is very easy to distinguish changes within the data. This means that the main goal of animating static data has been achieved in a much sleeker and easier to handle interface than that of multiple static graphs.

B. *Visualisation Advantages*

The advantages of this application over time based scatter plots or bubble charts and multiple line graphs can be divided into two different areas namely, visual advantages and informative advantages. These can be directly quantified with the positive feedback received from the user evaluation.

1) Visual advantages: These can be further broken down into scalability, and time and space based advantages. Scalable advantages can be attributed to the ability to scale and increase the view range to present all information to the user at once. Users said it was very easy to establish differences (due to the high contrast colours) and changes (due to use of animation) in the data. Users also noted that trends are easier to detect and understand when animated in the application, rather than scanning through static data. This can be attributed to the fact that multiple plots migrating across the stage quickly relay the trends they are exhibiting to the user.

Time and space-based advantages can be summed up as simply the amount of time and space saved when viewing the visualisation. Normally static graphs would require various graphs in order to visualise the same amount of data in order to be able to identify possible trends. Therefore time and space is saved by the ability to condense multiple static datasets into one fluid interface. Users said it would take them much less time to explore large datasets with our system than to keep track of not only trends in one graph but in many separate static graphs and plots voicing that they

instantly saw what the data was trying to convey.

2) Informative advantages: These can be seen as the ability to actually see movements in data as this is very useful in exploring trends. This is attributed to the animation of *x-coordinate*, *y-coordinate* and size values of ordered pair points. The ease of identifying these was cited as a big advantage by users trying to establish trends within the data. One problem which was raised by a notable portion of the user group was that they may need more time to initially learn and understand the system. This is in comparison to the static graphs which they have been exposed to for a much longer time.

V. CONCLUSION

The ability to input data into an application and have it dynamically generate a visual based interface was one of the main goals. This has been achieved through the use of Adobe Flash by creating an XML-based backend and vector-based frontend.

The animation of *x-coordinate*, *y-coordinate*, size and alpha (which creates the fade in effect) values led to a fluid environment where high contrast colours move across the graphing area, highlighting any trends within the data. The ability to map various trends in the form of their movement is not something that can be done when viewing static graphs. This was especially useful when using the application to view trends within the telecoms Internet data. Viewing all of this information in a static context would require many static graphs and charts to achieve the same level of information shown by the application. The user study conducted proved useful in both confirming our perceived advantages and raising some useful points in the area of application complexity.

This paper has presented a move towards a fluid data visualisation environment. The ability of the user to add their own data to the application allows dynamic data to be visualised. This means that nuances within any data can be identified more easily. The movement over time of values is no longer limited to interpretation of static graphs and hence is an improvement on existing ways of interpreting data.

A. Future Work

The application can be converted into a component which is an easy to setup and reusable object which can be inserted into any Flash application, movie or website. The authors are currently working on this extension.

VI. ACKNOWLEDGEMENTS

This work was undertaken in the Distributed Multimedia Centre of Excellence at Rhodes University, with financial support from Telkom, Comverse, Stortech, Tellabs, Amatole Telecom Services, Mars Technologies, Bright Ideas 39 and THRIP.

VII. REFERENCES

[1] Ronnlund, A R and Rosling, O. *Free Software for a World in Motion.* Proceeding of the Second International IEEE Conference on Creating, Connecting and Collaborating through Computing, 2004.

[2] Rosling, H and Ronnlund, A R. *New Software Brings Statistics Beyond the Eye. Statistics Knowledge and Policy: Key indicators to inform decision making*, OECD, 2006.

[3] Heer, J and Card, S K. *prefuse: a toolkit for interactive information visualization*. Human Factors in Computing Systems, University of California, Berkeley., 2005.

[4] Gapminder. Gapminder Foundation, 2008. [Online] Available: http://www.gapminder.org.

[5] Bilas, S. Advantages of Flash. Flash Magazine. [Online] Available: www.flashmagazine.com/tutorials/detail/advantages_of_flash/.

[6] Duignan, M and Biddle, R. *Evaluating Scalable Vector Graphics for use in Software Visualisation.* ACM International Conference, ACSI, 2003.

[7] Quin, L. Extensible Markup Language (XML). [Online] 2007. [Cited: April 24, 2009.] http://www.w3.org/XML.

[8] Delamothe, Tony. *TED 2006: The future we will create*. Vol. 332, 7540.

[9] Willuhn, D and Shulz, C. Developing accessible software for data visualization.. IBM Systems Jornal 32(4), 2003.

Simon Kerr completed his Bachelor of Science in 2006 majoring in Computer Science and Philosophy. He is currently reading towards a degree of Master of Science at Rhodes University.